CS241 #33 – Files, Filesystems #5

**> Processes (v2)**

virtual memory

threads, pid, ppid

open file descriptors (files,pipes,sockets)

uid, euid

pwd

meta information (e.g. total CPU time. Running status)

constraints (ulimits)

thread/process priority

umask

**> What is a process's umask and how is it used?**

What is the default value?

**> What is 'fstab' ?**

**> Case study: Use mount to explore a real filesystem**

**> Start a virtual machine using an iso file image**

qemu-system-x86\_64 -k en-us -cdrom dsl-4.4.10.iso

**> Welcome to the mmap diner. What would you like?**

void \*

**mmap**(void \*addr,size\_t len,int prot,int flags,int fd,off\_t off);

Gotcha: returns (void\*)-1 if failed

Ask yourself -

1. What kind of memory protection would you like?

2. Will the contents of your RAM (random access memory) be backed by a file or will be it anonymous?

3. What happens if you change your RAM contents? Will anyone know?

PROT\_EXEC ?

MAP\_SHARED or MAP\_PRIVATE. Choose one.

Got no file but still want to mmap? MAP\_ANONYMOUS!

Some demos -

fd = open("alice.txt", O\_RDONLY);

char \*ptr = mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_FILE|MAP\_SHARED, fd, 0);

if( (child=fork()) ==0)

strcpy (ptr, "The child wrote to the memory mapped area... ");

// what about the parent? What about the file?

fd2 = open("lyrics.txt", O\_RDONLY);

char \*ptr2 = mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_FILE|MAP\_PRIVATE, fd2, 0);

if( (child=fork()) ==0)

strcpy(ptr, "This is a private mapping");

**> What is RAID? Why is it necessary?**

Making filesystems resilient:

RAID

"Redundant Array of Inexpensive Disks"

RAID Motivation

Mean Time to Failure (MTTF)

MTTF (disk array) = MTTF (single disk) / # disks

Adding more disks means that failures happen more frequently..

Simplest form: Mirroring (also called “RAID 1”)

All data is mirrored across two disks

Advantages:

Reads are faster, since both disks can be read in parallel

Higher reliability (of course)

Disadvantages:

Writes are slightly slower, since OS must wait for both disks to do write

Doubles the cost of the storage system!

RAID 3

Rather than mirroring, use parity codes

Given N bits {b1, b2, ..., bN}, the parity bit P is the bit {0,1} that yields an even number of “1” bits in the set {b1, b2, ..., bN, P}

Idea: If any bit in {b1, b2, ..., bN} is lost, can use the remaining bits (plus P) to recover it.

Where to store the parity codes?

Add an extra “check disk” that stores parity bits for the data stored on

:

RAID 3 example

1. Read back data from other disks

2. Recalculate lost data from parity code

3. Rebuild data on lost disk

RAID 3 issues

Terminology

MTTF = mean time to failure

MTTR = mean time to repair

What is the MTTF of RAID?

Both RAID 1 and RAID 3 tolerate the failure of a single disk

RAID 5

Another approach: Interleaved check blocks (“RAID 5”)

Rotate the assignment of data blocks and check blocks across disks

Avoids the bottleneck of a single disk for storing check data

Allows multiple reads/writes to occur in parallel (since different disks affected)

Reliable distributed storage

Today, giant data stores distributed across 100s of thousands of disks across the world

e.g., your mail on gmail

**> A Planetary-sized Filesystem Case Study**

Problem: Build a file system for Google

How do you make it resilient?

Reliable distributed storage

Issues

Failure is the common case

Google reports 2-10% of disks fail per year

Now multiply that by 60,000+ disks in a single warehouse...

Must survive failure of not just a disk, but

failure of a rack of servers or even…

a whole data center

How:

GFS 2001: Simple redundancy (2 or 3 copies of each file)

GFS 2010:

More efficient redundancy (analogous to RAID 3++)

Reed-Solomon codes with 1.5x redundancy

RS codes found in CDs, Space communication protocols

Lots of interesting tidbits: http://goo.gl/LwFIy